Date=16/10/2020
Lecture By=Arkesh Jaiswal
Subject ⇒Javascript Events

IN PREVIOUS LECTURE **(QUICK RECAP) Date-15/10/2020**	**In Today's Lecture (Overview)**
• The HTML DOM (Document Object Model) • What is the HTML DOM? • The getElementById Method • The innerHTML Property • Finding HTML Elements • Changing HTML Elements • Adding and Deleting Elements • Finding HTML Objects • MCQ • Questions For Self practice / CC For The Day • Resource For The Lecture • Youtube Tutorial	• A series of fortunate events • Ways of using web events • Event handler properties • addEventListener() and removeEventListener() • MCQ'S • Questions For Self Practice / CC//Assignment For the Day • Resource For The Lecture • Youtube Tutorial

Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired. For example, if the user selects a button on a webpage, you might want to respond to that action by displaying an information box. In this article, we discuss some important concepts surrounding events, and look at how they work in browsers. This won't be an exhaustive study; just what you need to know at this stage

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, JavaScript first steps.
Objective:	To understand the fundamental theory of events, how they work in browsers, and how events may differ in different programming environments.

A series of fortunate events

As mentioned above, **events** are actions or occurrences that happen in the system you are programming — the system produces (or "fires") a signal of some kind when an event occurs, and provides a mechanism by which an action can be automatically taken (that is, some code running) when the event occurs. For example, in an airport, when the runway is clear for take off, a signal is communicated to the pilot. As a result, the plane can safely takeoff.

In the case of the Web, events are fired inside the browser window, and tend to be attached to a specific item that resides in it — this might be a single element, set of elements, the HTML document loaded in the current tab, or the entire browser window. There are many different types of events that can occur. For example:

- The user selects a certain element or hovers the cursor over a certain element.
- The user chooses a key on the keyboard.
- The user resizes or closes the browser window.
- A web page finishes loading.
- A form is submitted.
- A video is played, paused, or finishes.

- An error occurs.

A simple example

Let's look at a simple example of what we mean here. You've already seen events and event handlers used in many of the examples, but let's recap just to cement our knowledge. In the following example, we have a single `<button>`, which when pressed, makes the background change to a random color:

```
<button>Change color</button>
```

The JavaScript looks like so:

```
const btn = document.querySelector('button');

function random(number) {
  return Math.floor(Math.random() * (number+1));
}

btn.onclick = function() {
  const rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}
```

In this code, we store a reference to the button inside a constant called `btn`, using the `Document.querySelector()` function. We also define a function that returns a random number. The third part of the code is the event handler. The `btn` constant points to a `<button>` element, and this type of object has a number of events that can fire on it, and therefore, event handlers available. We are listening for the `click` event firing, by setting the `onclick` event handler property to equal an anonymous function containing code that generates a random RGB color and sets the `<body>` `background-color` equal to it.

This code is run whenever the click event fires on the `<button>` element, that is, whenever a user selects it.

The example output is as follows:

Ways of using web events

There are a number of ways to add event listener code to web pages so it runs when the associated event fires. In this section, we review the various mechanisms and discuss which ones you should use.

Event handler properties

These are the properties that exist to contain event handler code we have seen most frequently during the course. Returning to the above example:

```
const btn = document.querySelector('button');

btn.onclick = function() {
  const rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}
```

The `onclick` property is the event handler property being used in this situation. It is essentially a property like any other available on the button (e.g. `btn.textContent`, or `btn.style`), but it is a special type — when you set it to be equal to some code, that code is run when the event fires on the button.

You could also set the handler property to be equal to a named function name (like we saw in Build your own function). The following works just the same:

```
const btn = document.querySelector('button');

function bgChange() {
  const rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
random(255) + ')';
```

```
  document.body.style.backgroundColor = rndCol;
}

btn.onclick = bgChange;
```

addEventListener() and removeEventListener()

The newest type of event mechanism is defined in the Document Object Model (DOM) Level 2 Events Specification, which provides browsers with a new function — `addEventListener()`. This functions in a similar way to the event handler properties, but the syntax is obviously different. We could rewrite our random color example to look like this:

```
const btn = document.querySelector('button');

function bgChange() {
  const rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}

btn.addEventListener('click', bgChange);
```

Inside the `addEventListener()` function, we specify two parameters — the name of the event we want to register this handler for, and the code that comprises the handler function we want to run in response to it. Note: It is perfectly appropriate to put all the code inside the `addEventListener()` function, in an anonymous function, like this:

```
btn.addEventListener('click', function() {
    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
random(255) + ')';
    document.body.style.backgroundColor = rndCol;
  });
```

This mechanism has some advantages over the older mechanisms discussed earlier. First, there is a counterpart function, `removeEventListener()`, which removes a previously added listener. For example, this would remove the listener set in the first code block in this section:

```
btn.removeEventListener('click', bgChange);
```

Second, you can register multiple handlers for the same listener. The following two handlers wouldn't both be applied:

```
myElement.onclick = functionA; myElement.onclick = functionB;
```

The second line overwrites the value of `onclick` set by the first line. This would work, however:

```
myElement.addEventListener('click', functionA);
myElement.addEventListener('click', functionB);
```

MCQ'S

1.How do you create a new element in DOM?

A. document.create()
B. document.createChild()
C. document.createElement()

2.Which event is best suited for an autocomplete input?

A. click
B. change
C. Keyup

3.How do you capture the element triggering an event?

A. event.trigger
B. event.target
C. Event.element

4.How do you add event listeners?

A. addEventListener
B. addEventHandler

Questions For Self Practice / CC//Assignment For the Day

https://au-assignment.s3.ap-south-1.amazonaws.com/cc-82dab18f-26ef-4a89-a21f-c857314878ba.pdf
https://au-assignment.s3.ap-south-1.amazonaws.com/dom-a208f1de-d7f7-435a-90a7-050e4c7f5f75.pdf

Resource For The Lecture

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

Youtube Tutorial

https://www.youtube.com/watch?v=qS6OEzulgpc